iDyn

Serena Ivaldi
serena.ivaldi@iit.it

July 9, 2010- Draft version

Contents

Chapter 1

Introduction to iDyn

iDyn is a library for computing kinematics and dynamics of serial-links chains of revolute joints and **iCub** limbs. It started as an extension of the **iKin** library, with the purpose of including the dynamic description of the links, and gradually evolved until considering the whole body dynamics. Thus, starting from links (from iKinLink to iDynLink) the library provides classes for links, chain of links, and generic limbs. Of course, **iCub** limbs are included, with the specific kinematics and dynamics data already set (by default, the CAD parameters).

The library also provides a Newton-Euler based method to compute forces, moments, and joint torques, for each link in single or multiple interconnected chains, in presence or not of FT sensors, placed anywhere in the chain (e.g. in the middle of the chain).

1.1 Library modules

The library modules are:

- **RecursiveNewtonEuler** : force/torque computation in a chain using Newton-Euler recursive formulation, both in the forward/backward sense
- **iDynInv** : estimation of force/moment measured by a virtual FT sensor placed everywhere in a chain
- **iDynFwd** : retrieve joint torques of limbs with single FT sensor measurements
- **iDynBody** : whole body dynamics, i.e. interconnection of multiple chains
- **iFC** : projection of forces along the chain

1.2 Dynamics core

The base classes describing serial links chains are:

- iDynLink : a link with kinematic and dynamic characteristics
- iDynChain : a chain of serial links, with kinematic and dynamic properties
- iDynLimb : a generic limb, described by a chain

Note that iDynLink and iDynChain are derived from the corresponding classes (iKinLink and iKinChain) in the **iKin** library, so they provide both kinematic and dynamic data and methods. iDynLimb is, as in **iKin** , a redefinition of iDynChain methods, used to protect some chain data. iDynLink is an extension of iKinLink , since it adds the dynamic parameters of the link:

- H_C (HC,COM) : the center of mass, i.e. a roto-translational matrix defining the COM with respect to the link frame

- m (m): the link mass, concentrated in the COM
- I (I): the inertia matrix of the link

and the other characteristic variables used for dynamics:

- $\dot{q}$ (dq, dAng) : the joint velocity
- $\ddot{q}$ (ddq, d2q, d2Ang) : the joint acceleration
- w (w): link angular velocity
- $\dot{w}$ (dw): link angular acceleration
- $\ddot{p}$ (ddp): link linear acceleration
- $\ddot{p}_C$ (ddpC): the COM linear acceleration
- f (F): link force
- μ (M or Mu): link moment
- τ (Tau): joint torque

A iDynChain is basically a list of links; the main difference with respect to **iKin** is that blocked links contribute to the dynamics, so there's no need to have a second list to fasten dynamic computations. However, since iDynChain inherits from iKinChain , blocked links are considered for the Jacobian computation, which is unchanged. One of the advantages of **iDyn** is the possibility to interconnect multiple limbs, and iDynChain provides methods to compute a "shared" Jacobian: this topic will be discussed in the **iDynBody** module.

1.3 iCub limbs

Of course **iCub** parts/limbs (see Fig. 1.1) are already available to the user, pre-configured with the CAD parameters:

- iCubArmDyn : left/right arm of **iCub** (arm + torso, with blocked torso as in **iKin**)
- iCubArmNoTorsoDyn : left/right arm of the **iCub** (only arm)
- iCubTorsoDyn : torso of **iCub**
- iCubLegDyn : left/right leg of **iCub**
- iCubLegNoTorsoDyn : left/right leg of **iCub** without the torso roto-translation
- iCubEyeDyn : left/right eye of **iCub**
- iCubEyeNeckRefDyn : head of **iCub**
- iCubInertialSensorDyn : the inertial sensor in the **iCub** head
- iCubNeckInertialDyn : head of **iCub**

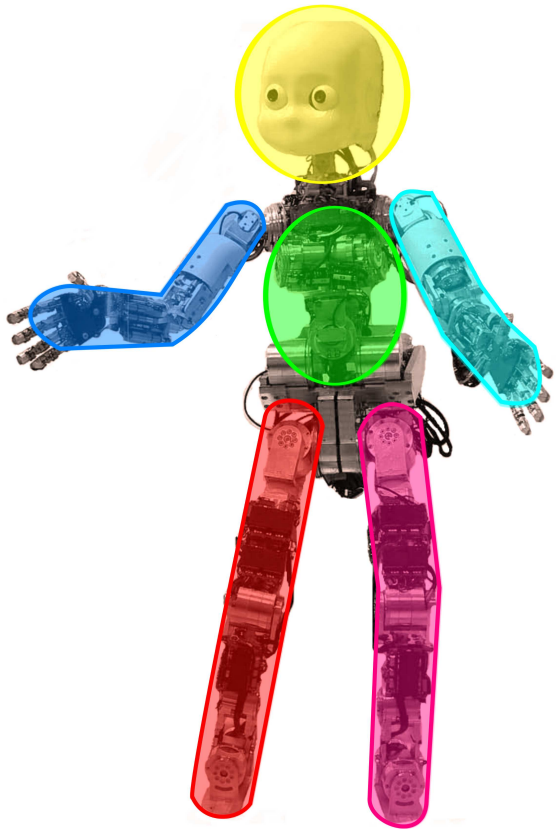

Figure 1.1: **iCub** 's whole body, with evidence of the limbs: head, left and right arm, torso, left and right leg.

Limb	N	DOF	Head	Torso	Arm	Leg	Eye
iCubArmDyn	10	7		012	3456789 0123456		
iCubArmNoTorsoDyn	7	7			0123456		
iCubTorsoDyn	3	3		012			
iCubLegDyn	6	6				012345	
iCubLegNoTorsoDyn	6	6				012345	
iCubEyeDyn	8	5	345 012	012			67 34
iCubEyeNeckRefDyn	5	5	012				34
iCubInertialSensorDyn	7	6	3456 345	012 012			
iCubNeckInertialDyn	4	3	3	012 012			

Table 1.1: iCub Limbs: the numbers $0-9$ refer to the joints index in each limb's link list.

Limb	H_0 —
iCubArmDyn	H0(0,1)=-1, H0(1,2)=-1, H0(2,0)=1, H0(3,3)=1
iCubArmNoTorsoDyn	H0 = eye
iCubTorsoDyn	H0 = eye
iCubLegDyn	H0 = eye, H0(2,3)=-0.1199, H0(1,3)=0.0681 (right) / -0.0681 (left)
iCubLegNoTorsoDyn	H0 = eye, H0(2,3)=-0.1199, H0(1,3)=0.0681 (right) / -0.0681 (left)
iCubEyeDyn	H0(0,1)=-1; H0(1,2)=-1; H0(2,0)=1; H0(3,3)=1;
iCubEyeNeckRefDyn	H0(0,1)=-1; H0(1,2)=-1; H0(2,0)=1; H0(3,3)=1;
iCubInertialSensorDyn	H0(0,1)=-1; H0(1,2)=-1; H0(2,0)=1; H0(3,3)=1;
iCubNeckInertialDyn	H0 = eye

Table 1.2: The base roto-translational matrix in the iCub Limbs

Chapter 2

RecursiveNewtonEuler

RecursiveNewtonEuler is a collection of base classes for computing forces/torques in one or multiple iDynChain , using Newton-Euler recursive formulation. The classes are:

- OneLinkNewtonEuler : a base class for computing forces and torques in a serial link chain
- BaseLinkNewtonEuler : a virtual base link
- FinalLinkNewtonEuler : a virtual final link
- SensorLinkNewtonEuler : a virtual sensor link
- OneChainNewtonEuler : a class for computing forces and torques in a iDynChain
- iCubArmSensorLink : a virtual sensor link on the **iCub** arm, for the arm FT sensor
- iCubLegSensorLink : a virtual sensor link on the **iCub** leg, for the leg FT sensor

2.1 Description

The main class which handles the computations of kinematic and wrench variables in the iDynChain is OneChainNewtonEuler , whereas OneLinkNewtonEuler is the base element class that provides all the Newton-Euler formulas to be applied between two iDynLink .

The kinematics variables are:

- w (w): link angular velocity
- $\dot{w}$ (dw): link angular acceleration
- $\ddot{p}$ (ddp): link linear acceleration

The wrench variables are:

- f (F): link force
- μ (M or Mu): link moment
- τ (Tau): joint torque

Since these quantities belong to the description of a iDynLink , each OneLinkNewtonEuler is basically a container of functions, pointing to its iDynLink to retrieve the data for the computations. A graphical representation is shown in Fig. 2.1.

OneChainNewtonEuler defines a chain of OneLinkNewtonEuler corresponding to an iDynChain : it constructs a chain with a Base, N OneLinks corresponding to the N links of the iDynChain , and a Final

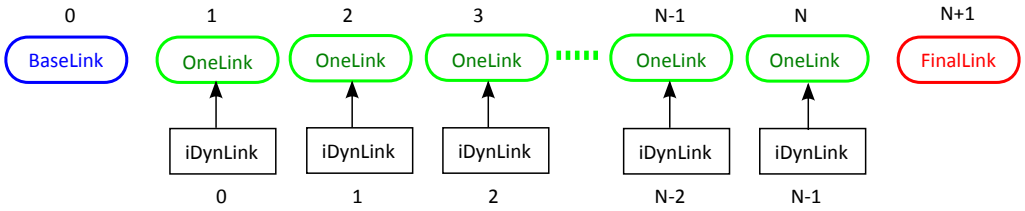

Figure 2.1: A OneChainNewtonEuler , consisting of a BaseLinkNewtonEuler , a FinalLinkNewtonEuler and many OneChainNewtonEuler , each pointing to its corresponding iDynLink in a iDynChain .

link. Differently from OneLinkNewtonEuler , BaseLinkNewtonEuler and FinalLinkNewtonEuler are "virtual" links, i.e. they do not have a corresponding iDynLink , but have the same features of a OneLinkNewtonEuler , in particular all the methods to be used for the application of Newton-Euler algorithm. It is important to stress that they are necessary for the correct application of the recursive computations, since they are used to initialize the two phases of the Newton-Euler algorithm. Moreover, as discussed later on, they will be necessary for the interconnection of multiple dynamic chains, where kinematic and wrench information are "shared" or "propagated" (see **iDynBody**).

The Newton-Euler algorithm consists of two phases:

- a **kinematic phase**, where the kinematic variables of the links are computed and "propagated" from the base link to the final link
- a **wrench phase**, where the wrench variables of the links are computed and "propagated" from the final link to the base link

In the classical formulation, the base link is a "virtual" link at the beginning of the manipulator chain (i.e. before the first link, or link $< 0 >$), whereas the final link is a "virtual" one at the end of the chain (usually the end-effector), so the kinematic variables are propagated "forward" (from base to end) and the wrench variables are propagated "backward" (from end to base). A graphical representation of the two phases is in Fig. 2.2.

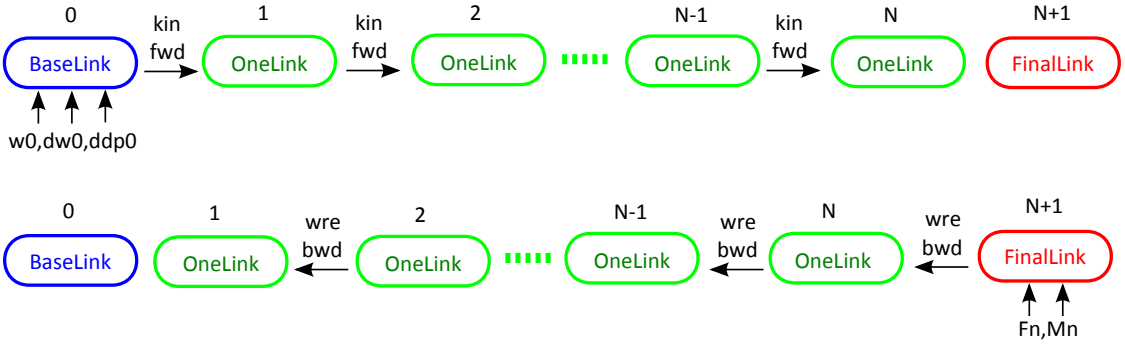

Figure 2.2: Kinematic and Wrench phase in the classical Newton-Euler algorithm, when solving a single dynamic chain.

Note that the distinction Base/Final is significant for the solution of a single chain, since it determines the starting point for the kinematics and wrench phase. The distinction is also related to the kinematics of the manipulator, since it follows the kinematics description of the manipulator, from a base (where a certain roto-translational matrix H_0 could be set) through all the links (defined by their Denavit-Hartenberg matrix) to the final link (usually, the end-effector where the most of the external interactions occurs). In particular, the kinematic variables are set in the Base, while the wrench in the Final link. They can be simply external measurements (e.g. a FT sensor is placed on the end-effector) or the result of some computations (e.g. from other chains).

2.2 A FT sensor in the chain

The FT sensor presence is crucial. If the chain is not provided with a FT sensor, a "virtual" sensor is assumed on the end-effector, and the external wrench is set to zero: this is, as an example, the case of a limb moving without constraints (collisions, interactions) in the space, when there are not external forces measured at the end-effector (or they are null). If a FT sensor is available, its measurements can be exploited to measure the external forces acting on the chain: if the sensor is on the end-effector, it naturally provides the wrench f_N, μ_N to initialize the wrench phase.

If the sensor is in a different position, i.e. in the middle of the chain, a SensorLinkNewtonEuler is created. This class defines the sensor frame with respect to the s-th physical link (iDynLink), where the sensor is attached to. Again, a "virtual" link is defined, having not a corresponding iDynLink ; but in this case it has a physical meaning because it represents the portion of the s-th link between the sensor and the s-th reference frame: so inertia, mass and COM for the "sub-link" are defined. Then this class can be integrated in the OneChainNewtonEuler as shown in Fig. 2.3.

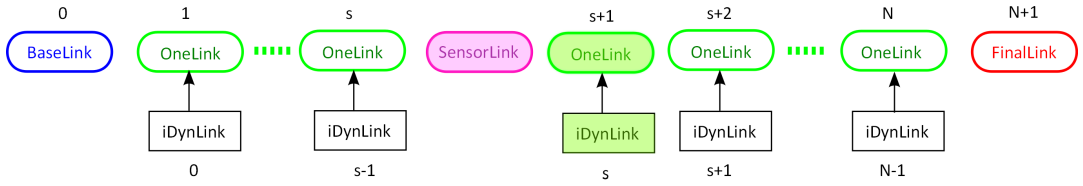

Figure 2.3: OneChainNewtonEuler where a FT sensor is in the middle of the chain, attached to the s-th iDynLink (highlighted in green).

In **iCub** limbs (both arms and legs), the FT sensor is instead placed in the middle of the chain: more precisely in the $2nd$ link of the arms ($5th$ if considering the arm with the torso) and in the $2nd$ of the legs. iCubArmSensorLink and iCubLegSensorLink are specific classes, with preset parameters of the sensor and its "sub-link". The unusual placement forced to design a different wrench phase in the Newton-Euler algorithm, which was adapted to the new situation, as shown in Fig. 2.4.

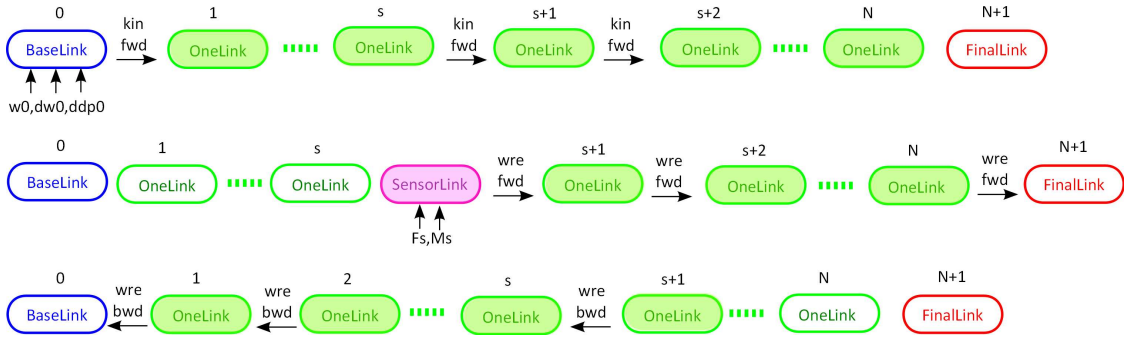

Figure 2.4: The Newton-Euler algorithm adapted to the case when a FT sensor is placed in the middle of the chain. In the second and third row the two parts of the wrench phase: the links involved in each part are highlighted in green.

The kinematic phase is unchanged: the kinematic information flows as in the "classic" case from the base up to the final link. The wrench phase is instead split into two:

- first, the wrench measured by the sensor is forwarded to the s-th link, in the so-called "attach": then the wrench information is propagated from the s-th link in the chain to the final link, in the so called "wrench-forward"
- second, the wrench information is propagated backward from the s-th link in the chain to the base link, in the so called "wrench-backward"

Chapter 3

iDynInv

iDynInv is a set of classes for computing links' forces, moments and joints torques in a kinematic chain where a single FT sensor is present, in any given position, and for also estimating its wrench given some external wrench. The sensor measurements are used to initialize the wrench phase of the recursive Newton-Euler computations in the whole chain. The classes are:

- iDynInvSensor : estimate the FT sensor wrench in a generic chain/limb
- iDynInvSensorArm : estimate the wrench for the FT sensor in a **iCub** arm (arm with blocked torso)
- iDynInvSensorArmNoTorso : estimate the wrench for the FT sensor in a **iCub** arm (arm only)
- iDynInvSensorLeg : estimate the wrench for the FT sensor in a **iCub** leg
- iDynSensor : compute joint torques given FT sensor measurements in a generic chain/limb
- iDynSensorArm : compute joint torques in a **iCub** arm (arm with blocked torso)
- iDynSensorArmNoTorso : compute joint torques in a **iCub** arm (arm only)
- iDynSensorLeg : compute joint torques in a **iCub** leg

3.1 Description

iDynInvSensor is a generic class, setting a FT sensor attached to a iDynChain (again, the sensor position in the chain is defined with respect to a certain link in the chain), by creating the corresponding SensorLinkNewtonEuler and "attaching" it into the OneChainNewtonEuler , as shown in Fig. 3.1.

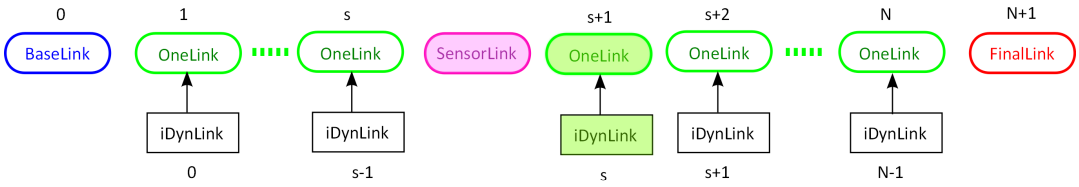

Figure 3.1: The FT sensor is placed in the middle of the chain, in the s-th iDynLink (highlighted in green). The corresponding SensorLinkNewtonEuler is then attached to the $s+1$-th OneLinkNewtonEuler in the corresponding OneChainNewtonEuler .

It is possible to compute an estimate of the FT sensor measurements by "attaching" the sensor to the s-th iDynLink and applying the wrench computation of Newton-Euler algorithm, specifically the "wrench-backward". The kinematic information are not necessary for the computation, so the kinematic phase is skipped. A graphical representation is shown in Fig. 3.2.

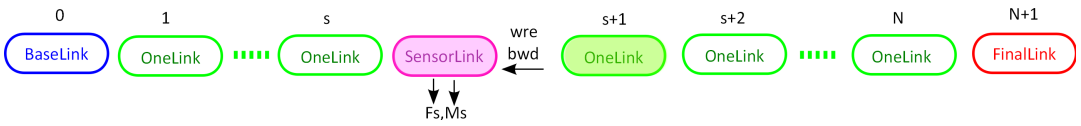

Figure 3.2: It is possible to estimate the FT sensor wrench by performing a wrench computation from the s-th iDynLink (the corresponding $s+1$-th OneLinkNewtonEuler is highlighted in green).

It is important to stress that iDynInvSensor assumes that the wrench of the s-th iDynLink has been already computed in some precedent computation: iDynInvSensor only performs the computation related to the sensor. While iDynInvSensor is generic for any limb, iDynInvSensorArm and iDynInvSensorLeg are specific classes performing computations for **iCub** arms and legs: by choosing left/right part (which is automatically set given the left/right arm/leg), they automatically set the proper FT sensor parameters, retrieved by CAD.

3.2 iDynFwd

iDynFwd is a set of classes for computing links' forces, moments and joints torques in a kinematic chain where a single FT sensor is present, in any given position. The sensor measurements are used to initialize the wrench phase of the recursive Newton-Euler computations in the whole chain. The classes are:

- iDynSensor : compute joint torques given FT sensor measurements in a generic chain/limb
- iDynSensorArm : compute joint torques in a **iCub** arm (arm with blocked torso)
- iDynSensorArmNoTorso : compute joint torques in a **iCub** arm (arm only)
- iDynSensorLeg : compute joint torques in a **iCub** leg

3.2.1 Description

iDynSensor is a generic class, which attaches a FT sensor into a iDynChain : the "attach" is again specified by the $s-th$ link hosting the sensor, the roto-translational matrix defining the sensor frame with respect to the link frame, and the dynamical parameters of the "sub-link" between the link frame and the sensor frame (the portion of link between the sensor and the end of the link). The corresponding SensorLinkNewtonEuler is created and to the limb's OneChainNewtonEuler . The corresponding SensorLinkNewtonEuler object, representing the "virtual" sub-link of the sensor, is created and "attached" to the OneChainNewtonEuler of the iDynChain , as shown in Fig. 3.3.

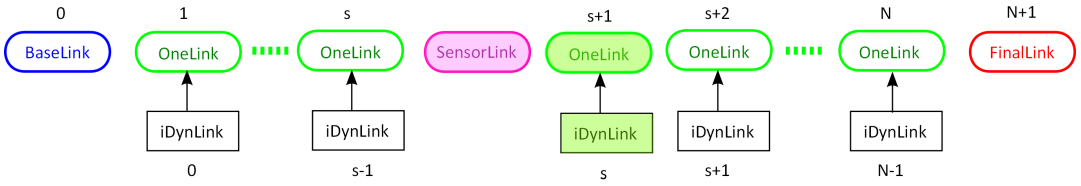

Figure 3.3: The FT sensor is placed in the middle of the chain, in the s-th iDynLink (highlighted in green). The corresponding SensorLinkNewtonEuler is then attached to the $s+1$-th OneLinkNewtonEuler in the corresponding OneChainNewtonEuler .

The FT sensor measurements can be then exploited to initialize the wrench phase of the Newton-Euler recursive algorithm, starting from the sensor instead of the end-effector of the chain (or the final link of the chain, represented by FinalLinkNewtonEuler). More precisely, the sensor wrench γ_s is "forwarded" to the $i-th$ link, then the wrench computations are split (see Fig. 3.4):

- from the $i-th$ link to the base of the chain (BaseLinkNewtonEuler), wrenches are computed with the classical backward formula
- from the $i-th$ link to the end-effector of the chain (FinalLinkNewtonEuler), wrenches are computed with the "inverse" forward formula

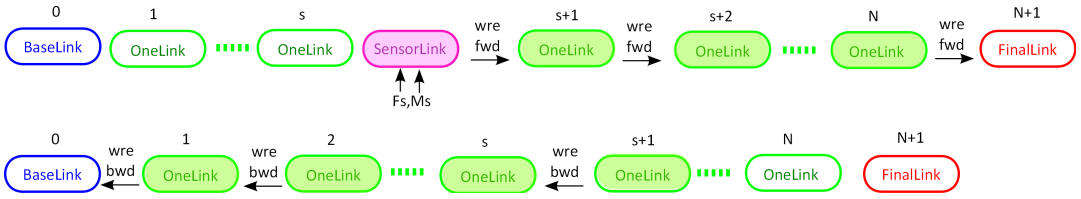

Figure 3.4: Given the FT sensor wrench measurements, the wrench phase is split.

The kinematics phase is precedent and the kinematics variables are "forwarded" from base to end, in the iDynChain , which provides all the interface methods to launch properly the phases in the correct order. It is interesting to point out that iDynSensor is derived from iDynInvSensor , so it also provided all the useful methods to estimate the sensor measurements given the external wrench acting on the end-effector of the chain.

iDynSensorArm , iDynSensorArmNoTorso and iDynSensorLeg , derived from iDynSensor , are specific for the **iCub** limbs which are provided with a FT sensor: legs and arms[1] In particular they are already "loaded" with the description of the specific FT sensors with respect to their limbs, so the user does not need to set the link index nor any other sensor parameter (mass, COM, etc.). The parameters have been retrieved by the **iCub** CAD design.

3.3 iDynBody

iDynBody contains a set of base classes for connecting multiple limbs and solve whole body dynamics. The classes are:

- RigidBodyTransformation : a class for connecting a limb to a node
- iDynNode : the base class defining a connection between/among multiple limbs
- iDynSensorNode : a node where one or multiple limbs have a FT sensor
- iDynSensorTorsoNode : a specific structured node, used to derive **iCub** upper and lower torso
- iCubUpperTorso : head, left and right arm
- iCubLowerTorso : torso, left and right leg
- iCubWholeBody : head, torso, and both left and right arms and legs

3.3.1 Description

iDynNode represents a virtual node where multiple limbs are connected, and may exchange kinematic and wrench information among them. Multiple limbs can be attached to the Node, but at least one must be attached.

[1]There are two different **iDyn** classes for defining **iCub** 's arms: the first is inherited from **iKin** , meaning that the chain of the arm also contains the 3 torso links, which are blocked (blocked links are not counted in the DOF but are necessary for the correct computation of the Jacobian, referred to the base of the torso); the second only contains the true arm links and does not have the torso ones. Since users acquainted with **iKin** may prefer that convention, both options have been considered. The difference between iDynSensorArm and iDynSensorArmNoTorso is the index of the link whom the sensor is attached to.

The mutual exchange between node and limbs (full duplex) is managed used a RigidBodyTransformation class, containing the roto-translational matrix which describes the connection, and the type of "flow" of kinematic and wrench variables: from limb to node (**RBT_NODE_IN**) or from node to limb (**RBT_NODE_OUT**). One limb can be attached to a node at the base or at the end-effector: this, combined with the information flows, affects the computations in particular the application of the Newton-Euler algorithm while solving the limbs dynamic. The node is empty at its creations. When the limbs are "added" to the node, a progressive number is assigned to each limb, with its RigidBodyTransformation : this number is the limb ID, and is used to address the limbs correctly when setting external informations, e.g. measured forces.

If one or multiple limbs have a FT sensor, the node evolves in a iDynSensorNode , which also keeps track of the iDynSensor 's of each limb: again, the limb progressive ID number is used to match each couple limb-sensor.

Both iDynNode and iDynSensorNode are generic base classes: one can start from them to build any kind of robot, as the interconnection of multiple nodes and multiple limbs. An example of use of the two nodes is presented in Fig. 3.5: then in Fig. 3.6 an example of flow for iDynNode is shown, while in Fig. 3.7 for iDynSensorNode .

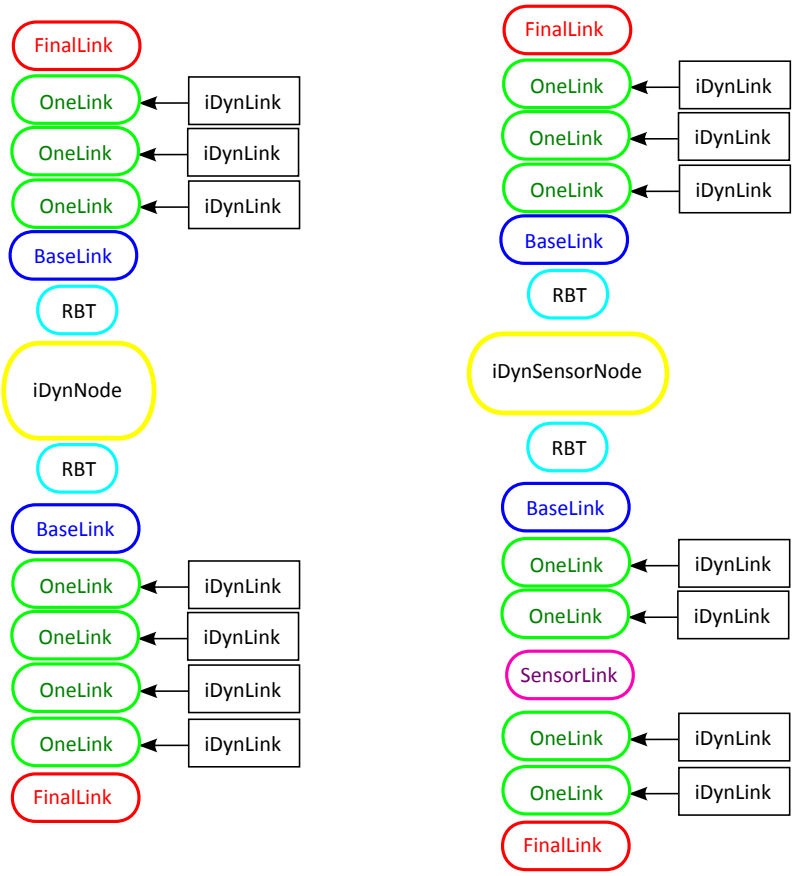

Figure 3.5: Examples of iDynNode and iDynSensorNode , with a 3DOF and a 4DOF limbs attached.

3.3.2 iCub's whole body

A particular type of iDynSensorNode is the iDynSensorTorsoNode : it connects a central-up limb, a left and right limb (called *up*, *right* and *left*), when both left/right limb have FT sensors and the central-up one uses the kinematic and wrench information coming from some external input (a sensor or a iDynSensorNode). This class has been created as it is the base for iCubUpperTorso and iCubLowerTorso , which have some

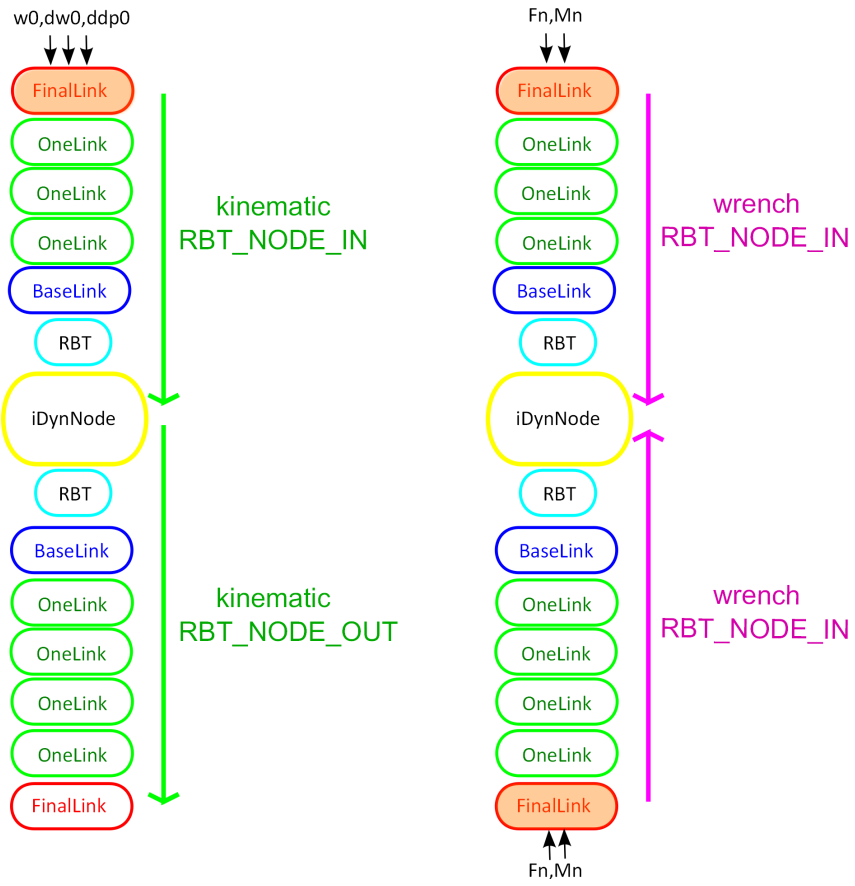

Figure 3.6: The flow of kinematic and wrench information in a typical iDynNode .

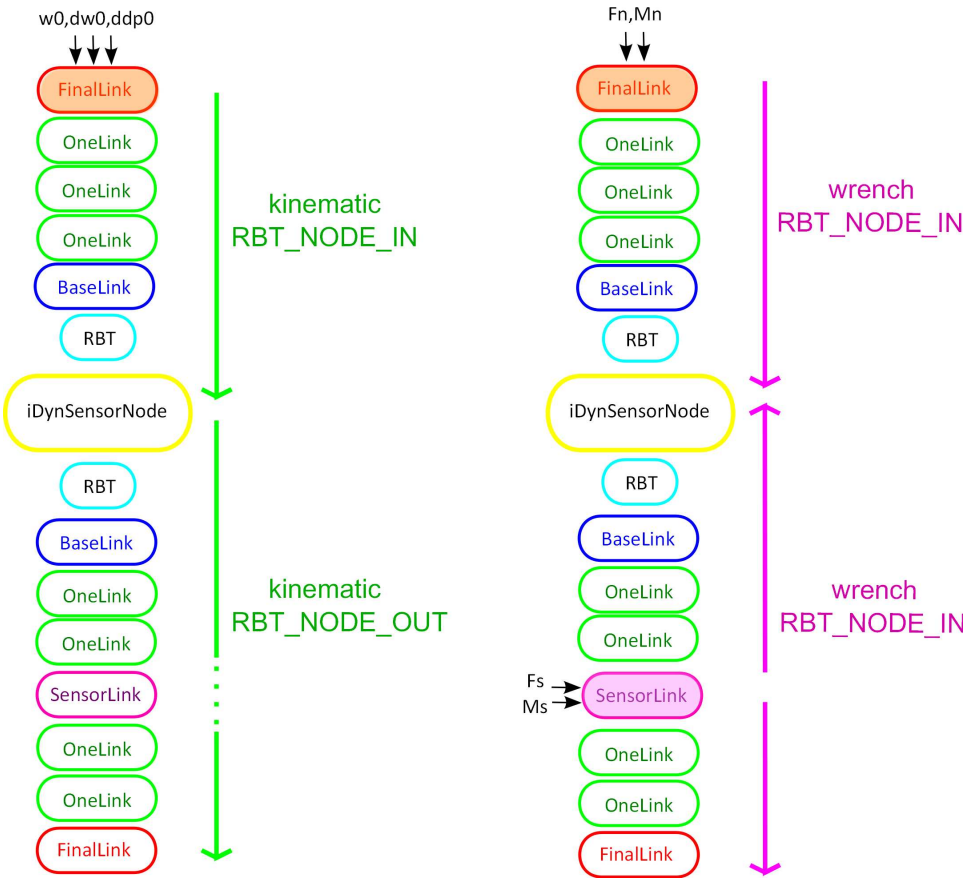

Figure 3.7: The flow of kinematic and wrench information in a typical iDynSensorNode . Note that the kinematic information is not propagated in the sensors.

structural analogies. In fact, **iCub** 's body can be described by the connection of two big nodes: the **upper body**, consisting of head, right and left arm, and the **lower body**, consisting of torso, right and left arm. In both cases, the limbs are attached to the node via their base, and the information flows are the same: legs and arms are identical from a conceptual point of view, since hands and feet are end-effectors, and all have a FT sensor in the middle of the chain. Head and torso are also similar, because both have external kinematic and wrench inbound information (the inertial sensor in the head, and the upper node for the torso). Hence, the two nodes share the same structure.

iCubWholeBody is the ultimate class, describing **iCub** in terms of connection between upper and lower body: the upper torso is connected to the lower through a RigidBodyTransformation , connecting the iCubUpperTorso with the final link of the iCubTorsoDyn chain, in iCubLowerTorso . The the whole body can be accessed. iCubWholeBody s merely a container: pointers to upper and lower torso objects are accessible, so all public methods of the two objects can be used.

Node	**Limb**	H_0	H_{RBT}
UpperTorso	iCubArmNoTorsoDyn R	eye	$\begin{bmatrix} -0.9659 & 0 & 0.2588 & 0.0031 \\ 0 & 1 & 0 & -0.0500 \\ -0.2588 & 0 & -0.9659 & -0.1103 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
UpperTorso	iCubArmNoTorsoDyn L	eye	$\begin{bmatrix} 0.9659 & 0 & -0.2588 & 0.0029 \\ 0 & -1 & 0 & -0.0500 \\ -0.2588 & 0 & -0.9659 & 0.1100 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
UpperTorso	iCubNeckInertialDyn	eye	eye
UpperTorso	iCubTorsoDyn	eye	eye
LowerTorso	iCubTorsoDyn	eye	eye
LowerTorso	iCubLegNoTorsoDyn R	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.0681 \\ 0 & 0 & 1 & -0.1199 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 & -0.1199 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & -0.0681 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
LowerTorso	iCubLegNoTorsoDyn L	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -0.0681 \\ 0 & 0 & 1 & -0.1199 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 & -0.1199 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 0.0681 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Table 3.1: The RBT roto-translational matrix for the **iCub** whole body.

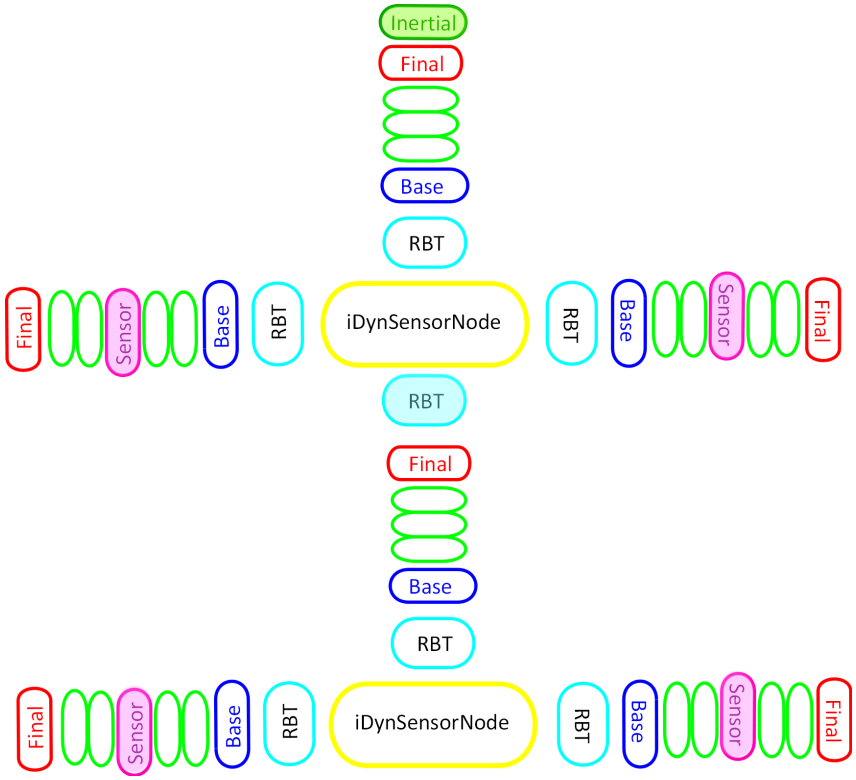

Figure 3.8: The scheme of **iCub** 's whole body, as the connection between two big iDynSensorNode : the iCubUpperTorso , consisting of head and arms, and the iCubLowerTorso , consisting of torso and legs. The RigidBodyTransformation connecting the upper and the lower torso via the torso limb is highlighted in cyan. The available sensors are also put in evidence: the inertial sensor (green) on top of the head, which is the source of kinematic information; and the FT sensor on arms and legs (pink), which are the sources of wrench information.

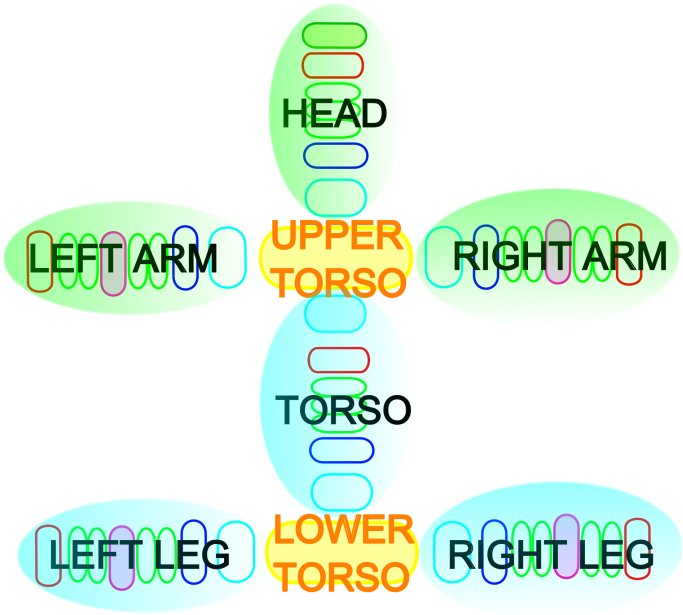

Figure 3.9: **iCub** 's whole body, with evidence of nodes and limbs.

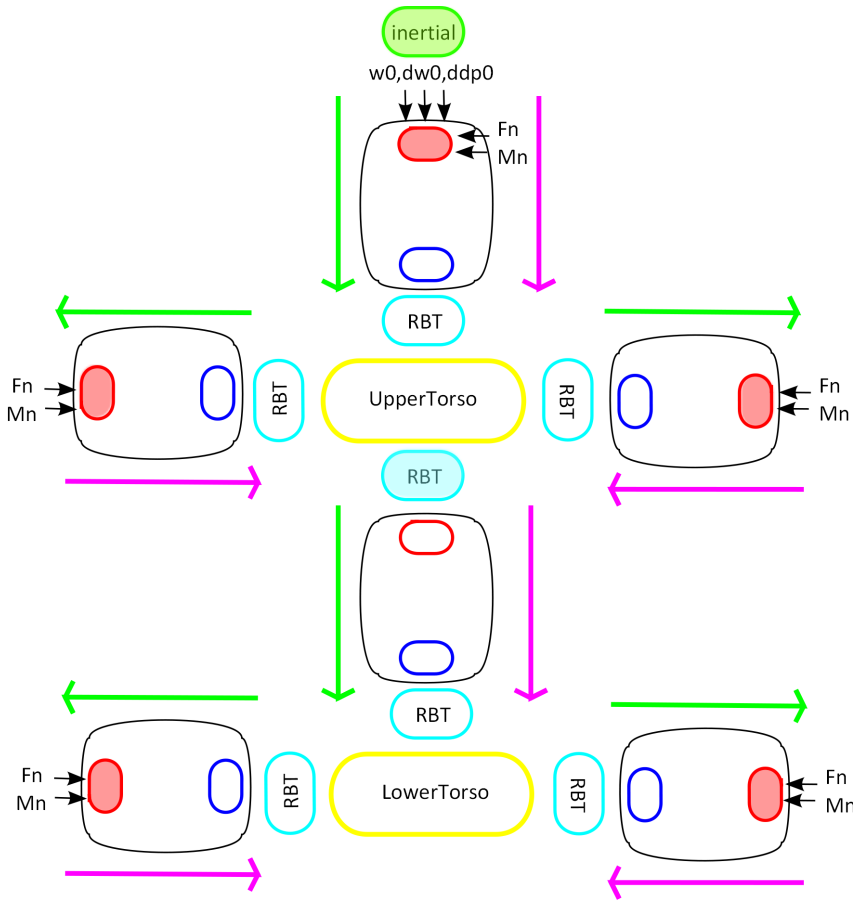

Figure 3.10: The flow of kinematic and wrench information in **iCub** whole body, when only the inertial sensor information is available (highlighted in green) and no FT sensor on arms/legs are available: in this case, arms and legs in the wrench phase are initialized with the "external" wrench, acting on the end-effector, and set in the final link. If there are FT sensors on these end-effectors, their measurements can be used; otherwise the external wrench is null by default. The kinematic flow is the green arrow, the wrench flow is the magenta arrow.

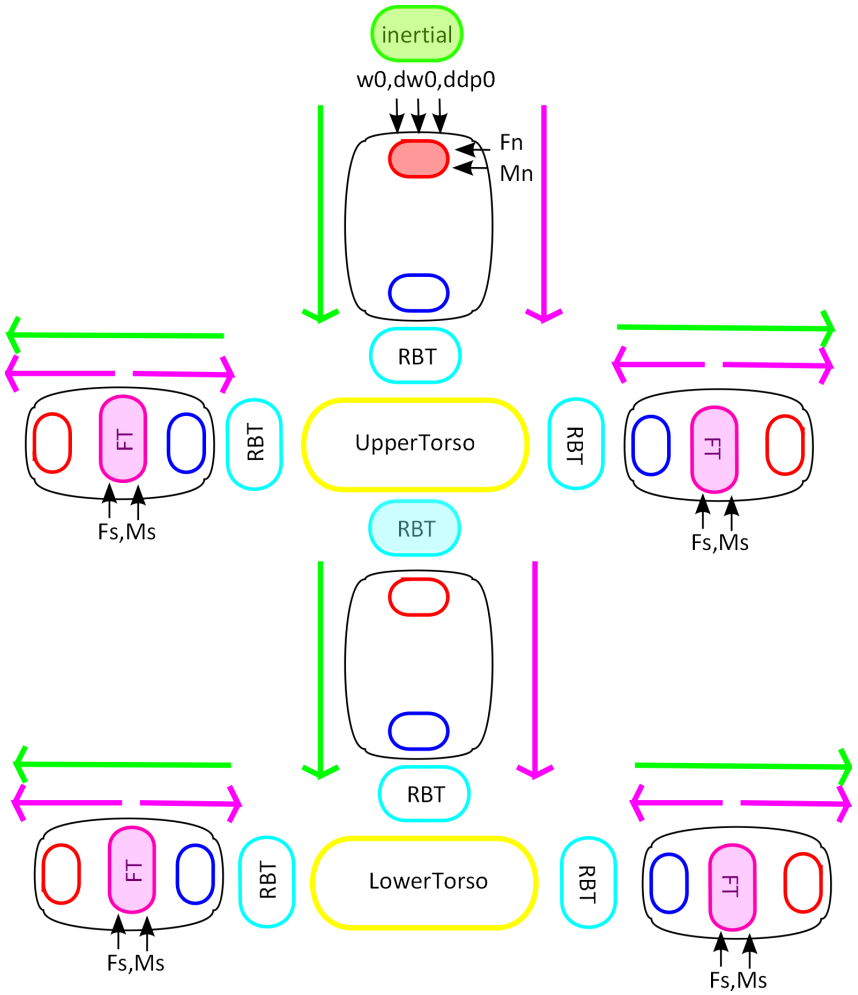

Figure 3.11: The flow of kinematic and wrench information in **iCub** whole body, when the inertial sensor information is available (highlighted in green) and FT sensor on arms/legs are available: in this case, arms and legs in the wrench phase are initialized with the sensor measurements, and the wrench phase (split into two parts) is performed by a proper iDynSensor . The head wrench is instead null by default. The kinematic flow is the green arrow, the wrench flow is the magenta arrow.